Programming - Part I

Ricard Solé Casas

June 15, 2017

Foreword

The source code for this report and app can be found on Github[1]. The live version of the app itself is online at http://quiz.rsole.me.

Some of the decisions taken in building this app do not follow the original suggested guidelines. The UI is a web frontend, but all the business logic is handled by the server-side via Java.

Declaration

I confirm that the submitted coursework is my own work and that all material attributed to others (whether published or unpublished) has been clearly identified and fully acknowledged and referred to original sources. I agree that the College has the right to submit my work to the plagiarism detection service. TurnitinUK for originality checks.

Acknowledgements

I'd like to thank my partner Shannon for her continued support and challenges that help me grow, both professionally and personally. I would also like to thank all of you who also helped me get here.

[1] https://github.com/rcsole/coursework-java

Contents

Chapter 1

Design Choices

I've made some choices for this solution that require some justification. The project guidelines suggested using the Java Swing[1] framework, which was meant to be replaced by JavaFX[2]. However, both technologies are quite outdated. The former is partially deprecated and the latter doesn't appear to be well maintained, documentation is scarce, and Oracle has discontinued support for the editor tools. After some research I came to the conclusion that, as of June 2017, the community has pivoted towards using Java for the business logic and persistence through a web server, while leaving the templating and UI sections to a much more mature and seasoned technology: the HTML, CSS, and JavaScript triad.

In this particular project, the server is built on the shoulders of the Java bindings to the Play[3] framework (see footnote for more information). The persistence layer is built on Java's Ebean[4] targeting a PostgreSQL[5] backend. For deployment, I used Heroku[6] and integrated it with Github[7] for automated deployments.

I've also decided against displaying which questions were answered incorrectly. The reason for that is that it would serve as *cheating*. One would just have to retake the entire Quiz and know which ones they have to actually change. Hiding which answers were incorrectly answered makes, in my opinion, for a more interesting game overall.

Likewise I've also decided against prompting the user when skipping a question. I think that prompting the user **every time** they want to skip ahead involves too many clicks.

In the following chapter I will elaborate on the core pieces of the application.

[1] https://www.wikiwand.com/en/Swing_(Java)
[2] https://www.wikiwand.com/en/JavaFX
[3] https://playframework.com
[4] http://ebean-orm.github.io/
[5] https://www.postgresql.org/
[6] https://heroku.com
[7] http://www.giphy.com/gifs/3ohzdEZt9v5mq8oAsE

Chapter 2

Overview

Project is setup using the Java Play[1] framework to aid with following the MVC[2] pattern.

2.1 Models

From Wikipedia:

> *The model is the central component of the pattern. It expresses the application's behavior in terms of the problem domain, independent of the user interface. It directly manages the data, logic and rules of the application.*

All the models used by this application are stored in PostgreSQL[3]. The bindings are done through PlayEbean[4].

2.1.1 Option

`Option` is the smallest model. It has a one-to-many[5] relationship to `Question`, where one `Question` can have `n Option`s, and each `Option` belongs to `1 Question`.

In essence it's an alias to a `String` type, the only difference is that `Option` has a `Long id` and a `Question question`.

See models/Option.java[6] for implementation.

2.1.2 Question

`Question` is, arguably, the meat of the application. It holds *many* `Option`s as a `List<Option>`, along with other parameters like difficulty, type, and category or the `Quiz` it belongs to. Like `Option`, `Question` holds a one-to-many[7] relationship to `Quiz`. Except in this case `n Question`s belong to `1 Quiz`.

See models/Question.java[8] for implementation.

[1] https://playframework.com
[2] https://www.wikiwand.com/en/Model%E2%80%93view%E2%80%93controller
[3] https://www.postgresql.org/
[4] https://www.playframework.com/documentation/2.5.x/JavaEbean
[5] https://www.wikiwand.com/en/One-to-many_(data_model)
[6] https://git.io/vHdem
[7] https://www.wikiwand.com/en/One-to-many_(data_model)
[8] https://git.io/vHde6

2.1.3 Quiz

This is the model that is actually used by other parts of the application. The rest are intermediary models to store data in a way that makes it easier to manipulate. `Quiz` owns two different models, `QuizResult` and `Question`. Any `Quiz` may have any number of `QuizResult`s and `Question`s.

It also has a `String difficulty` value which ranges from *easy* to *hard*, or *mixed*. Also provides a method int `computeScore(DynamicForm answers)` to compute the score from a form submission.

See models/Quiz.java[9] for implementation.

2.1.4 QuizResult

`QuizResult` is how the application stores the return value of `int computeScore(DynamicForm answers)` in `Quiz`. This is to allow sharing and retaking the `Quiz`, hence the belonging relationship.

See models/QuizResult.java[10] for implementation.

2.2 Services

From SWE SX[11]:

> *Services normally return DTOs in large applications or domain models directly in smaller applications. DTOs normally means more work, but better separation of concerns. The typical flow is: Controller calls service → Service returns an object (be it a DTO, domain model or something else) → Controller maps DTO/domain model to a view model.*

2.2.1 QuizService

The service layer in this case abstracts the added complexity of calling the Open Trivia DB[12] API. It makes a request to the API with modifiers such as `String amount` and `String difficulty`, and then returns a `Quiz` Ebean.

See services/QuizService.java[13] for implementation.

2.3 Controllers

From Wikipedia:

> *The controller accepts input and converts it to commands for the model or view.*

2.3.1 QuizResultsController

This controller handles requests that ask for a specific result, as well as creating new results. The former is via the `Result show(UUID id)` method; the latter is via the `Result create()` method.

See controllers/QuizResultsController.java[14] for implementation.

[9] https://git.io/vHdvZ
[10] https://git.io/vHdvn
[11] https://softwareengineering.stackexchange.com/a/211724
[12] https://opentdb.com/
[13] https://git.io/vHdkc
[14] https://git.io/vHdks

2.3.2 QuizzesController

This controller is the meatiest one. It renders a form with `Result form()`, handles the result of said form in `CompletionStage<Result> create()` and displays an existing `Quiz` via `Result show(UUID id)`.

See controllers/QuizzesController.java[15] for implementation.

[15] https://git.io/vHdkn

Chapter 3

Critical Evaluation

This application could be improved in the following aspects:

- `ManyToMany` relationship between `Quiz` and `Question` to avoid duplicated `Question` entries.
- Display difficulty per question.
- Keyboard shortcuts for UI.
- Variable time length depending on the difficulty of each question.

Chapter 4

Test plan

4.1 Manual tests

Test	*Method*	*Expected*	*Actual*	*Evidence*
Selecting 10 creates a quiz with 10 questions	Click 10	There will be 10 questions	As expected	See figure 4.1
Selecting 20 creates a quiz with 20 questions	Click 20	There will be 20 questions	As expected	See figure 4.2
Selecting 30 creates a quiz with 30 questions	Click 30	There will be 30 questions	As expected	See figure 4.3
Cross on top left takes user back to quiz creation	Click x	Quiz will go back to form	As expected	See gif[1]
Selecting an option brings the next question up	Select an option	Next question will come up	As expected	See gif[2]
Skipping will send the question to the end	Skip a question	The question will be skipped and asked again at the end of the quiz	As expected	See gif[3]
When the timer runs out the quiz gets submitted	Wait for timer to run out	The quiz gets submitted	As expected	See gif[4]
Score is displayed as a percentage	Finish the quiz	The score is displayed upon finishing the quiz	As expected	See figure 4.4

[1] http://www.giphy.com/gifs/3ohzdEZt9v5mq8oAsE
[2] http://www.giphy.com/gifs/3og0IMCTcnr7RFvaaQ
[3] http://www.giphy.com/gifs/l1BgSVSrual0DUzDi
[4] http://www.giphy.com/gifs/l0Iy8yTqqq5BCf1yU

4.2 Automated tests

See https://github.com/rcsole/coursework-java/tree/master/test for a far more complete suite of tests.

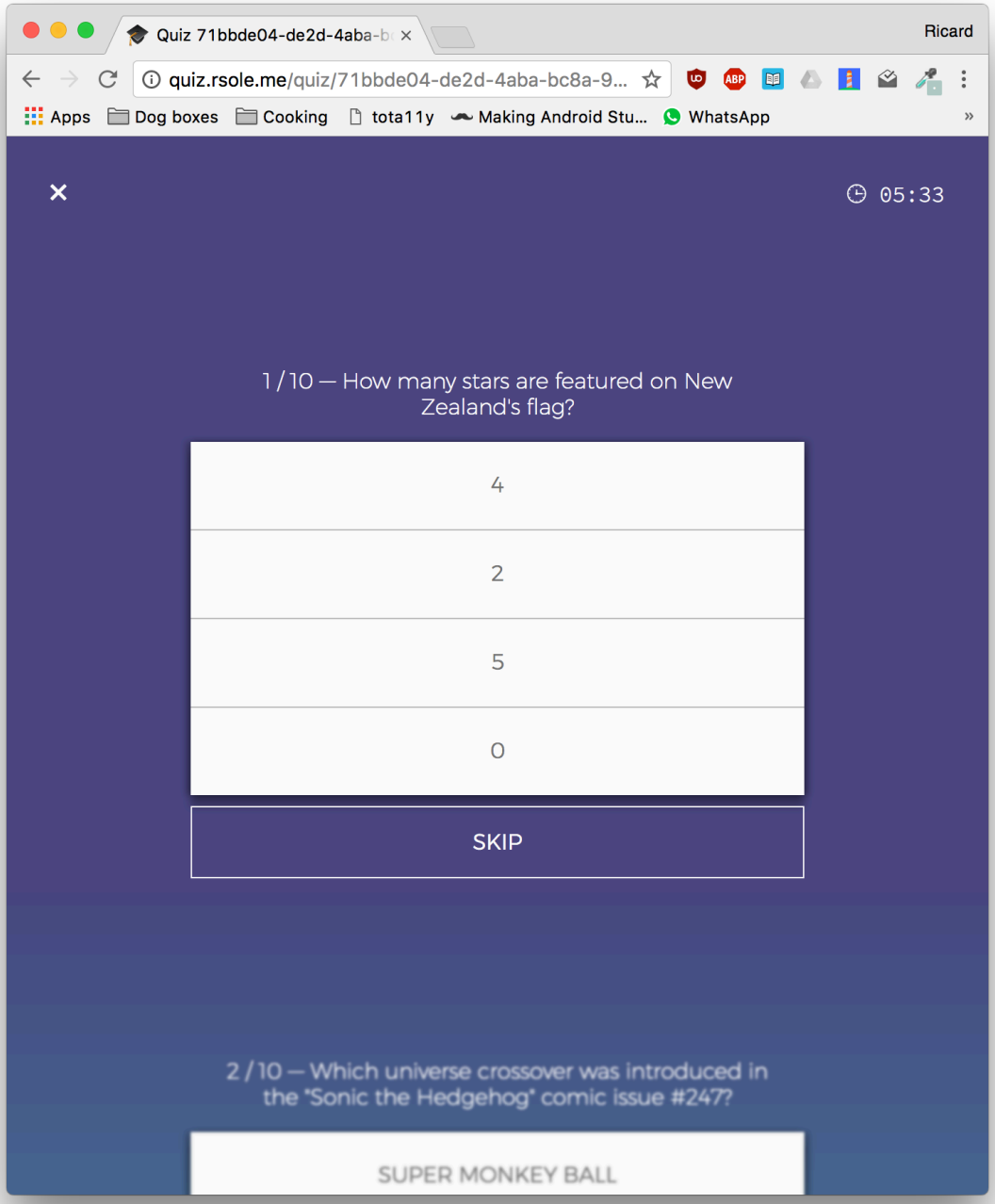

Figure 4.1: 10 Questions

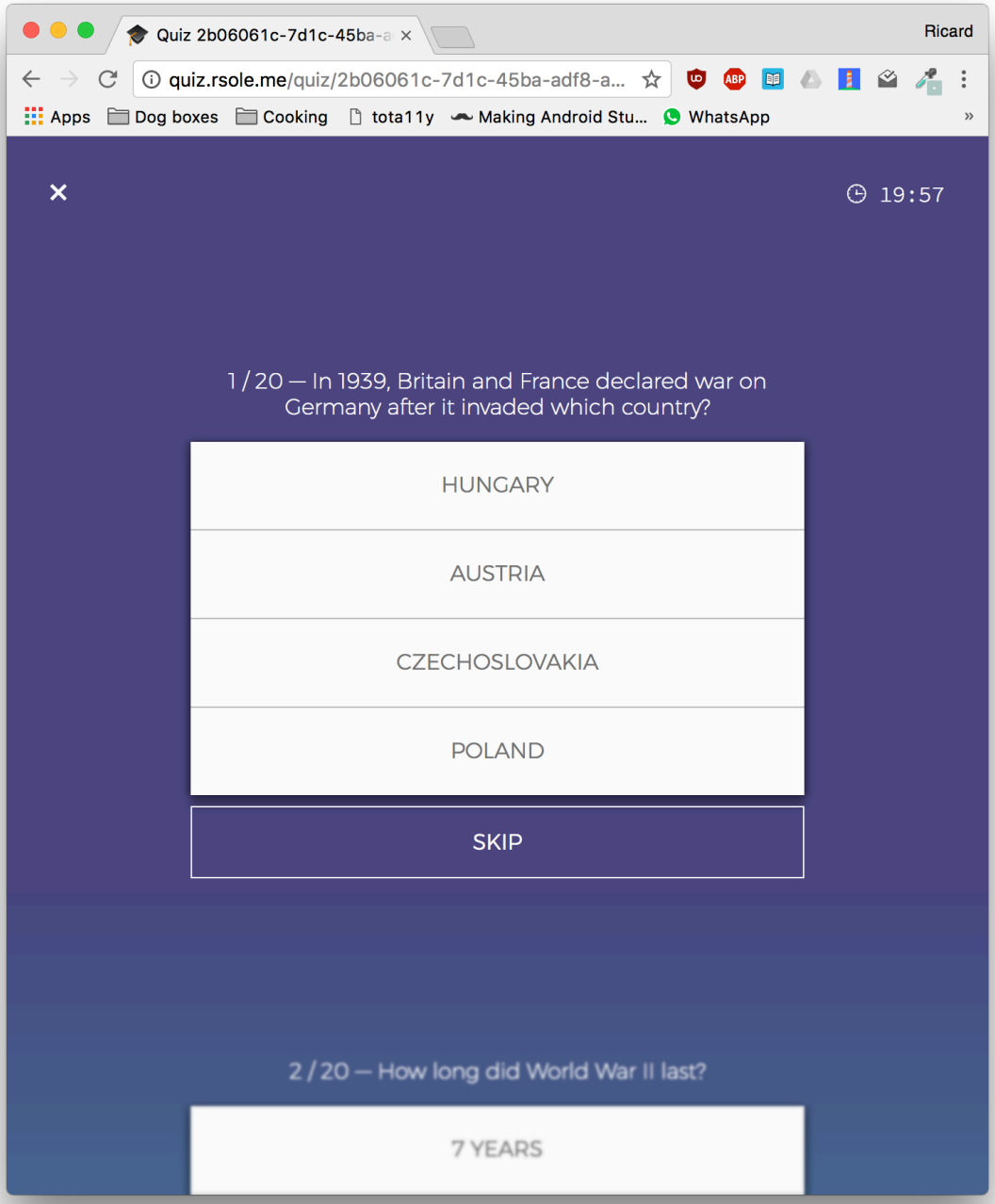

Figure 4.2: 20 Questions

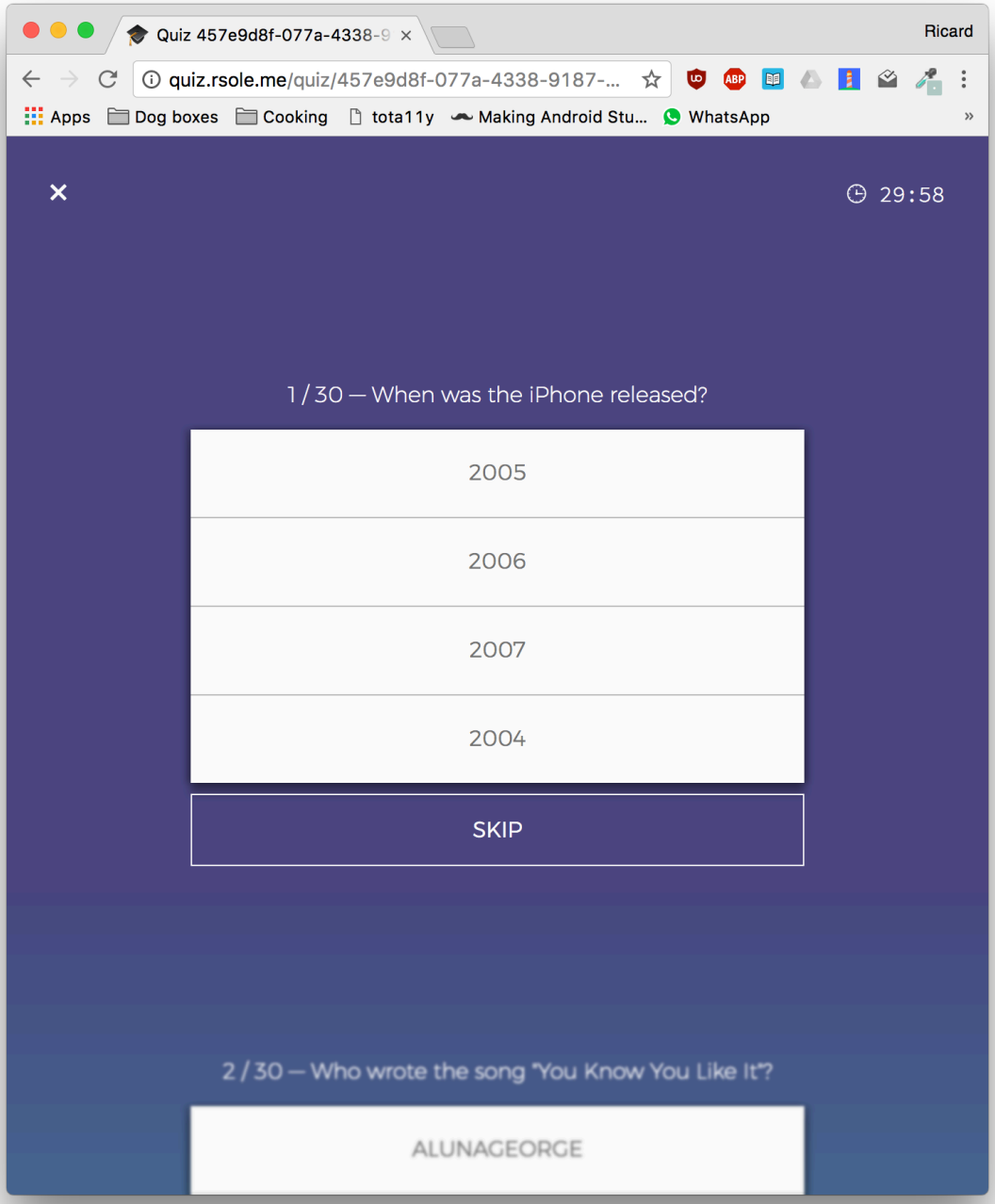

Figure 4.3: 30 Questions

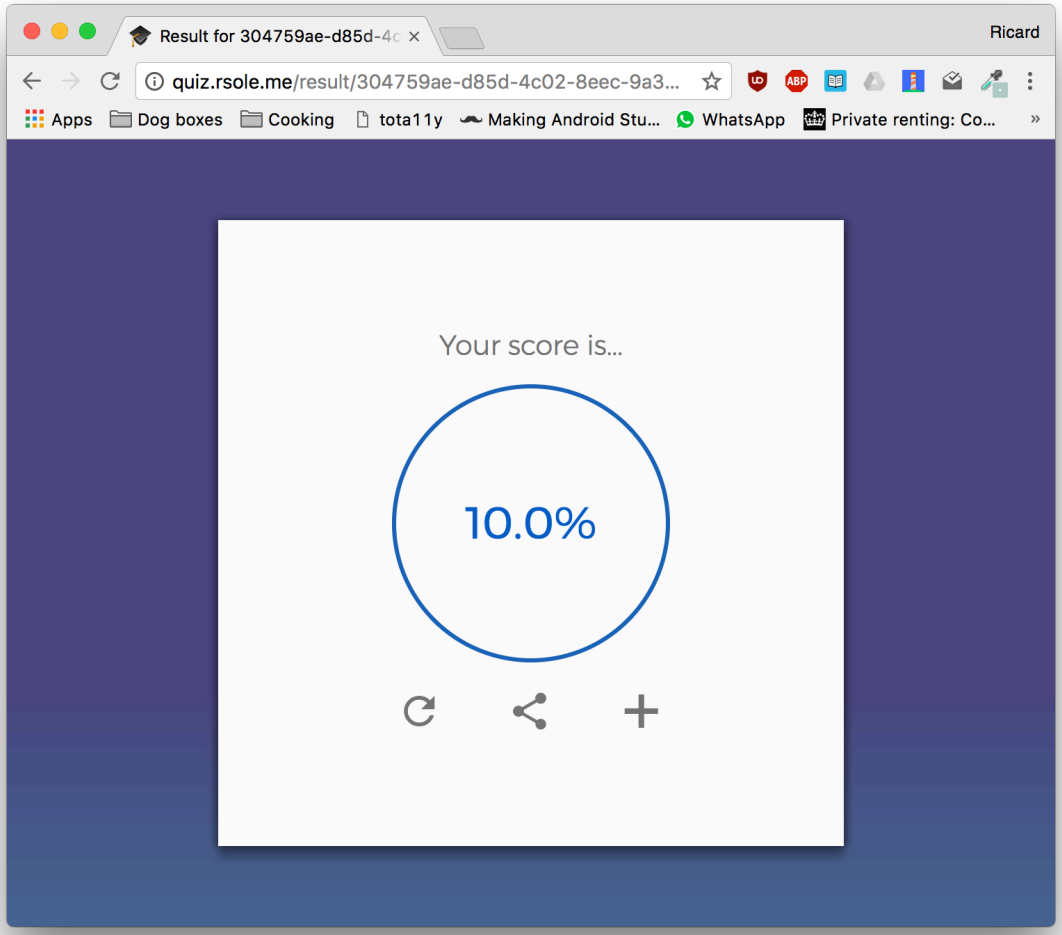

Figure 4.4: Quiz result

Chapter 5

Appendix A: Source Code

5.1 QuizResultsController.java

```
package controllers;

import com.avaje.ebean.Ebean;
import com.google.inject.Inject;
import models.Quiz;
import models.QuizResult;
import play.data.DynamicForm;
import play.data.FormFactory;
import play.mvc.Controller;
import play.mvc.Result;

import java.util.UUID;

public class QuizResultsController extends Controller {
  @Inject private FormFactory formFactory;

  public Result create() {
    DynamicForm requestData = formFactory.form().bindFromRequest();
    Quiz q = Ebean.find(Quiz.class, UUID.fromString(requestData.get("quiz-id")));
    QuizResult r = new QuizResult();
    r.setScore(q.computeScore(requestData));
    r.setQuiz(q);
    r.save();

    return redirect("/result/" + r.getId());
  }

  public Result show(UUID id) {
    QuizResult r = Ebean.find(QuizResult.class, id);

    return ok(views.html.results.show.render(r));
  }
}
```

5.2 QuizzesController.java

```
package controllers;

import com.avaje.ebean.Ebean;
import com.google.inject.Inject;
import models.Quiz;
import play.data.DynamicForm;
import play.data.FormFactory;
import play.mvc.Controller;
import play.mvc.Result;
import services.QuizService;
import views.html.quizzes.form;
import views.html.quizzes.show;

import java.util.UUID;
import java.util.concurrent.CompletionStage;

public class QuizzesController extends Controller {
  @Inject private FormFactory formFactory;
  private QuizService service = new QuizService();

  public Result form() {
    return ok(form.render());
  }

  public CompletionStage<Result> create() {
    DynamicForm requestData = formFactory.form().bindFromRequest();
    String amount = requestData.get("questionsAmount");
    String difficulty = requestData.get("difficulty");

    return service
        .fetch(amount, difficulty)
        .thenApply(
            (Quiz q) -> {
              if (q.getQuestions().size() == 0) return redirect("/");

              return redirect("/quiz/" + q.getId());
            });
  }

  public Result show(UUID id) {
    Quiz q = Ebean.find(Quiz.class, id);

    return ok(show.render(q));
  }
}
```

5.3 Option.java

```
package models;

import javax.persistence.*;

@Entity
@Table(name = "options")
```

```
public class Option {
  @Id private Long id;
  private String text;

  @ManyToOne(cascade = CascadeType.ALL)
  private Question question;

  public Option(String t) {
    this.text = t;
  }

  public Long getId() {
    return id;
  }

  public void setId(Long id) {
    this.id = id;
  }

  public String getText() {
    return text;
  }

  public void setText(String text) {
    this.text = text;
  }

  public Question getQuestion() {
    return question;
  }

  public void setQuestion(Question question) {
    this.question = question;
  }

  @Override
  public String toString() {
    return "Option{" + "text='" + text + '\'' + '}';
  }
}
```

5.4 Question.java

```
package models;

import com.avaje.ebean.Model;
import com.fasterxml.jackson.annotation.JsonProperty;

import javax.persistence.*;
import java.util.ArrayList;
import java.util.Collections;
import java.util.List;
import java.util.UUID;

@Entity
@Table(name = "questions")
public class Question extends Model {
```

```
@Id private UUID id;

@ManyToOne(cascade = CascadeType.ALL)
private Quiz quiz;

@JsonProperty("question")
private String text;

private String category;
private String type;
private String difficulty;

@JsonProperty("correct_answer")
private String correctAnswer;

@JsonProperty("incorrect_answers")
@OneToMany(cascade = CascadeType.ALL, mappedBy = "question")
private List<Option> incorrectAnswers;

public UUID getId() {
  return id;
}

public void setId(UUID id) {
  this.id = id;
}

public Quiz getQuiz() {
  return quiz;
}

public void setQuiz(Quiz quiz) {
  this.quiz = quiz;
}

public String getText() {
  return text;
}

public void setText(String text) {
  this.text = text;
}

public String getCategory() {
  return category.split("\\s|:")[0].toLowerCase();
}

public void setCategory(String category) {
  this.category = category;
}

public String getType() {
  return type;
}

public void setType(String type) {
  this.type = type;
}
```

```
  public String getDifficulty() {
    return difficulty;
  }

  public void setDifficulty(String difficulty) {
    this.difficulty = difficulty;
  }

  public String getCorrectAnswer() {
    return correctAnswer;
  }

  public void setCorrectAnswer(String correctAnswer) {
    this.correctAnswer = correctAnswer;
  }

  public List<Option> getIncorrectAnswers() {
    return incorrectAnswers;
  }

  public void setIncorrectAnswers(List<Option> incorrectAnswers) {
    this.incorrectAnswers = incorrectAnswers;
  }

  public List<Option> getOptions() {
    List<Option> os = new ArrayList<>(this.incorrectAnswers);
    os.add(new Option(this.correctAnswer));
    Collections.shuffle(os);

    return os;
  }
}
```

5.5 Quiz.java

```
package models;

import com.avaje.ebean.Model;
import com.fasterxml.jackson.annotation.JsonIgnoreProperties;
import com.fasterxml.jackson.annotation.JsonProperty;
import play.data.DynamicForm;

import javax.persistence.*;
import java.util.List;
import java.util.UUID;

@Entity
@Table(name = "quizzes")
@JsonIgnoreProperties({"response_code"})
public class Quiz extends Model {
  @Id private UUID id;

  @OneToMany(mappedBy = "quiz", cascade = CascadeType.ALL)
  @JsonProperty("results")
  private List<Question> questions;
```

```
  @OneToMany(mappedBy = "quiz", cascade = CascadeType.ALL)
  private List<QuizResult> quizResults;

  private String difficulty;

  public UUID getId() {
    return id;
  }

  public void setId(UUID id) {
    this.id = id;
  }

  public List<Question> getQuestions() {
    return questions;
  }

  public void setQuestions(List<Question> questions) {
    this.questions = questions;
  }

  public String getDifficulty() {
    return difficulty;
  }

  public void setDifficulty(String difficulty) {
    this.difficulty = difficulty;
  }

  public int computeScore(DynamicForm answers) {
    int score = 0;

    for (Question q : questions) {
      String a = answers.get(q.getId().toString());
      if (a != null && a.equals(q.getCorrectAnswer())) score += 1;
    }

    return score;
  }

  public List<QuizResult> getQuizResults() {
    return quizResults;
  }

  public void setQuizResults(List<QuizResult> quizResults) {
    this.quizResults = quizResults;
  }
}
```

5.6 QuizResult.java

```
package models;

import com.avaje.ebean.Model;

import javax.persistence.*;
import java.util.UUID;
```

```
@Entity
@Table(name = "quiz_results")
public class QuizResult extends Model {
  @Id private UUID id;

  @ManyToOne(cascade = CascadeType.PERSIST)
  private Quiz quiz;

  private int score;

  public UUID getId() {
    return id;
  }

  public void setId(UUID id) {
    this.id = id;
  }

  public Quiz getQuiz() {
    return quiz;
  }

  public void setQuiz(Quiz quiz) {
    this.quiz = quiz;
  }

  public int getScore() {
    return score;
  }

  public double getPercentage() {
    return ((double) score / (double) quiz.getQuestions().size()) * 100;
  }

  public void setScore(int score) {
    this.score = score;
  }
}
```

5.7 QuizService.java

```
package services;

import com.fasterxml.jackson.databind.ObjectMapper;
import models.Quiz;
import play.libs.ws.WS;
import play.libs.ws.WSRequest;
import play.libs.ws.WSResponse;

import java.io.IOException;
import java.util.concurrent.CompletionStage;

public class QuizService {
  public CompletionStage<Quiz> fetch(String amount, String difficulty) {
    return request(amount, difficulty)
        .thenApply((WSResponse r) -> r.asJson().toString())
```

```
        .thenApply((String json) -> save(json, difficulty));
  }

  private CompletionStage<WSResponse> request(String amount, String difficulty) {
    final String HOST = "https://opentdb.com/api.php";
    final WSRequest req =
        WS.url(HOST)
            .setQueryParameter("amount", amount)
            .setQueryParameter("type", "multiple")
            .setContentType("application/json");

    if (!difficulty.equals("mixed")) {
      return req.setQueryParameter("difficulty", difficulty).get();
    }

    return req.get();
  }

  private Quiz save(String json, String difficulty) {
    ObjectMapper m = new ObjectMapper();
    try {
      Quiz q = m.readValue(json, Quiz.class);
      q.setDifficulty(difficulty);
      q.save();
      return q;
    } catch (IOException e) {
      return new Quiz();
    }
  }
}
```